Luis Garcia Remes

Game AI - Assignment 2

First Steps

The first graph that I created is based off of my house. I made a node to represent each room or area in or around my house and used a rough estimate of the distance between the center of each area to its neighboring areas as the cost of each edge. I ended up with a graph of 20 nodes and 40 edges.

The second graph I used is one of Rome, Italy that I found on the website that you had suggested in class. The nodes represent road intersections and the edges are the roads themselves. I chose it because it was the smallest data set they had on the site. I wanted to use a large graph but the other ones seemed too large to actually be useful for this assignment.

I created a method to parse a graph in csv form into my DirectedGraph class. All the graphs I used can be found inside my project file under the DirectedGraphs directory. Each line in the csv represents an edge with its source node as the first value, the sink as the second value, and the cost of the edge as the third value.

Dijkstra's Algorithm, A*

As far as graph structure performance, I used std::vector for pretty much every struct or class that I used so I definitely went for the make it work first mentality and didn't worry about being efficient. As far as runtime efficiency, based on the tests that I have run the more nodes that your search visits the more time it takes to find the desired path (Image E). The amount of memory used by my algorithm also goes up the more nodes need to be visited since every time you visit a new node I have to create additional node records for them. The small graph was too small to have any observable difference in time between starting the search and when it ends (at least when measuring time in milliseconds).

Heuristics

I used both constant guess and random guess for running A* on both my large and small graphs. For the random guess I had a random number assigned to the edge

between 1 and 50 for the small graph and 1 and 500 for the large graph. For the constant guess I just used the total number of nodes for each graph as the estimated value for each edge in that graph. I also used a Euclidean distance heuristic for the grid part of the assignment that I talk more about in the section below.

I found that the different heuristics didn't change the number of nodes visited or the path found for the small graph. However I found that the random guess yielded less nodes visited but a longer path than using the constant guess which means that my random guess was likely overestimating and therefore is not admissible and not consistent. I am guessing that the constant value I used was underestimating the actual value and therefore it is admissible and consistent. I also found that both versions of A* that I ran yielded more nodes visited but shorter paths (Image D) than the path generated by Dijkstra's.

Putting it All Together

For the last part of the assignment I decided to use a grid division scheme as you had suggested in class. I created a csv file after designing the obstacle course in google sheets. I decided that all spaces in a grid should have edges between itself and its surrounding neighbors that did not have an obstacle contained in it. I chose this because I figured this would allow me to have my boid maneuver closer to the obstacles (which seems more believable to me if you are trying to find the shortest path) without having to worry about obstacle avoidance. I used the arrive and look where you are going algorithms from the first assignment (after fixing the issues that they had) for the path navigating part of the pathfinding algorithm. For the heuristic in this part of the assignment I decided to use Euclidean distance since I could easily calculate it for each and every node based on the clicked node during runtime (I would imagine that it would be better in an actual game to have all this precomputed to save on speed and processing power).

When I first started debugging the pathfinding I found that sometimes the boid would start to follow its path only to stop half way through the list of nodes. I checked to see if the issue was my logic but I couldn't figure out what was going wrong until I started printing the distance between the boid and the current node it was pathfinding to. It seemed like for some specific paths the boid would reach a point where it would never reach the distance it needed to switch to the next target. I realized that I had made the target radius in the arrive algorithm larger than the threshold distance for the switch. What I still don't understand is why this was causing an issue for a few specific paths and not every path.

I also found a weird quirk that happens when you select a goal node that is one space offset from the starting node and some straight line distance from it in the other axis (See Images A and B). In my mind the most reasonable path for the boid to take in image A would be to go all the way to the left until it is under the left most obstacle in the row containing the target and then move diagonally up and left into the target node. For image B the most reasonable path for the boid to take would be to move all in a straight line up and then go diagonally left and up from the node to the right of the upper most obstacle in the column containing the target. Instead the boid decides to take a wavy path to the target. These wavy paths aren't necessarily any more costly than the alternative paths that I expected (at least I didn't think it should be considering the cost I was using for each edge going from a node to its accessible neighbors is 1) because the number of nodes that the boid needs to arrive to are the same. I assumed that this issue was occuring because the cost of an edge between a node and its horizontal and vertical neighbors was the same as the cost for its diagonal neighbors. So I tried changing the cost of diagonal edges to cost twice as much as the cost of vertical and horizontal neighbors and ran a few more examples (Image C). It turns out that the change I made did absolutely nothing which made me realize (at least I think) that the issue stems from the fact that I am using a Euclidean distance as a heuristic. Because the heuristic makes the search biased towards nodes that are in the "right direction' it prioritizes diagonal movements over orthogonal movements. In Image A the distance between the node to the left of the starting node has a distance of sqrt((7)^2+(1)^2) while the node it selected has a distance of sqrt((7)^2+(0)^2) so it prioritizes the diagonal movement although it wouldn't make sense from a human perspective to go towards more obstacles when the path ahead is completely clear (unless maybe the boid is getting shot at from somewhere below its starting node's row and its trying to stay behind cover in an fps).

The last interesting thing I noticed when putting it all together was that I would sometimes have an issue where the boid would decide to clip into an obstacle when trying to avoid it. I first assumed the issue was due to pathing errors but after adding the path printing function it became apparent that the boid was being told to avoid spaces with obstacles but was still trying to "cut through" them as it traversed its intended path. After a bit of messing around with the values of my arrive function I found that if the max linear acceleration of the boid was set too low it wouldn't be able to make sharp turns when switching from one target to the next which would manifest itself in the boid lazily changing directions and clipping into spaces it shouldn't be walking into. After upping the linear acceleration the clipping issues appear to have gone away but just as a precaution I made the obstacles I tiny bit smaller than the space they are in

Appendix

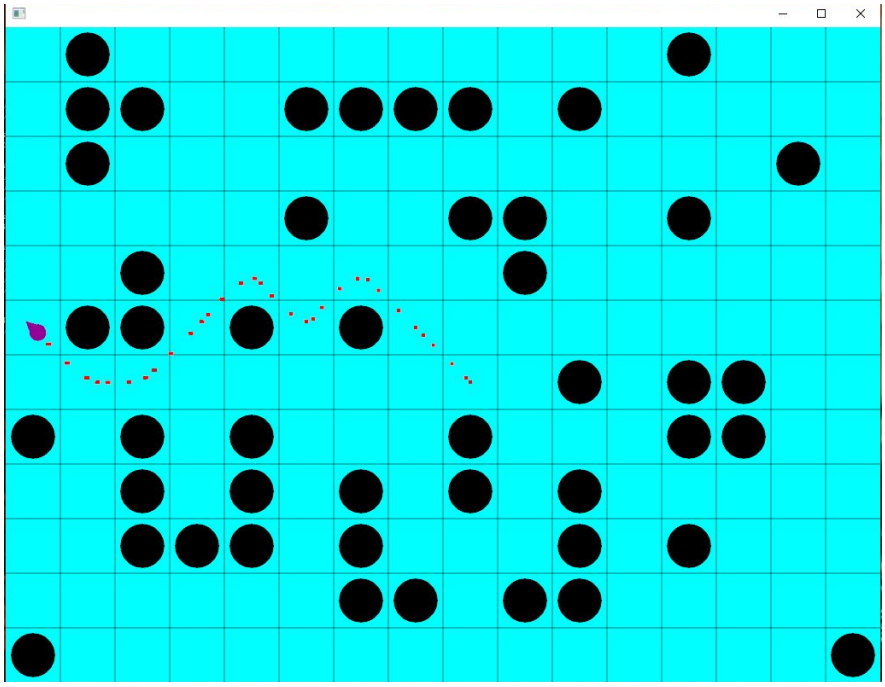

Image A: Path Finding ZigZag 1

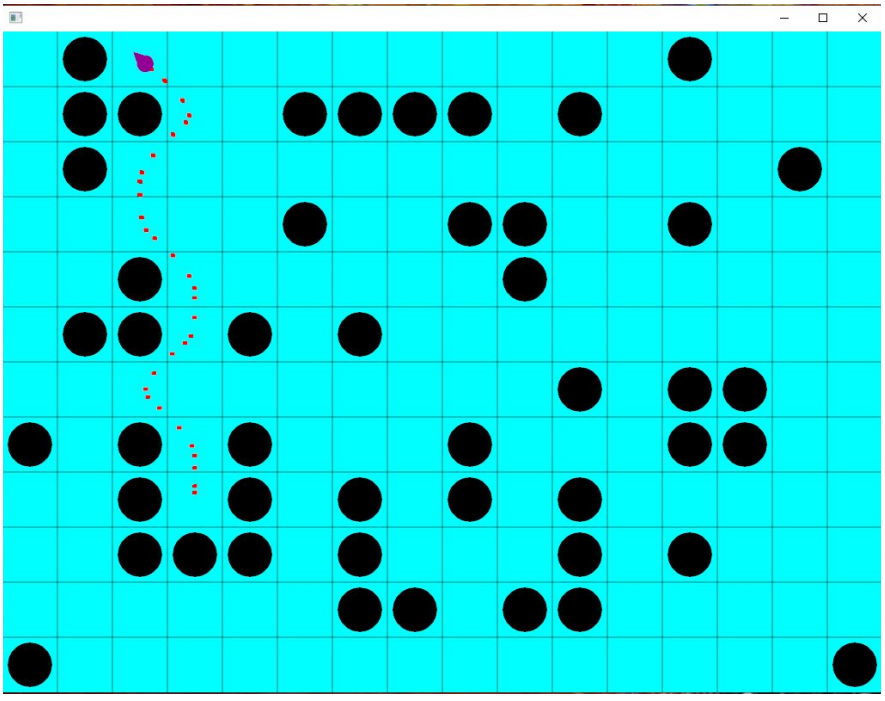

Image B: Path Finding ZigZag 2

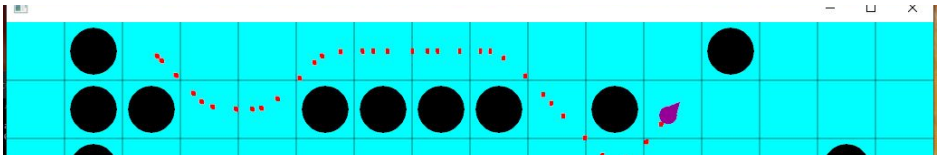

Image C: Path Finding ZigZag 3 - Diagonal More Expensive

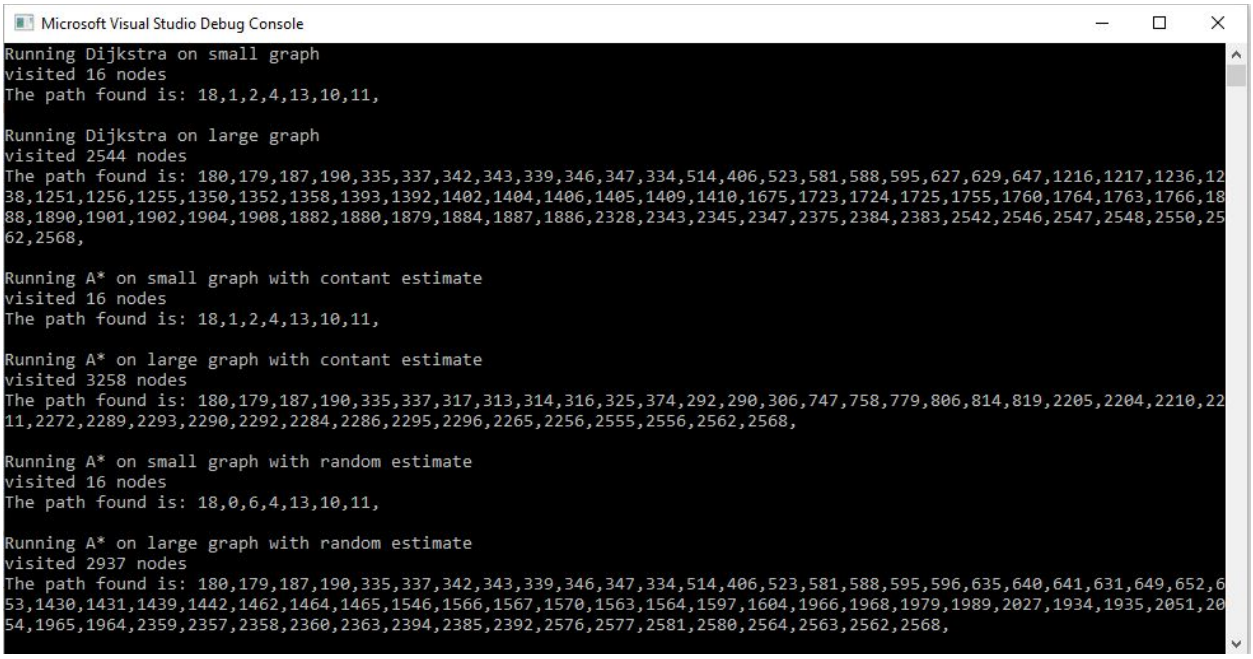

Image D: Dijkstra and A* on small and large graphs

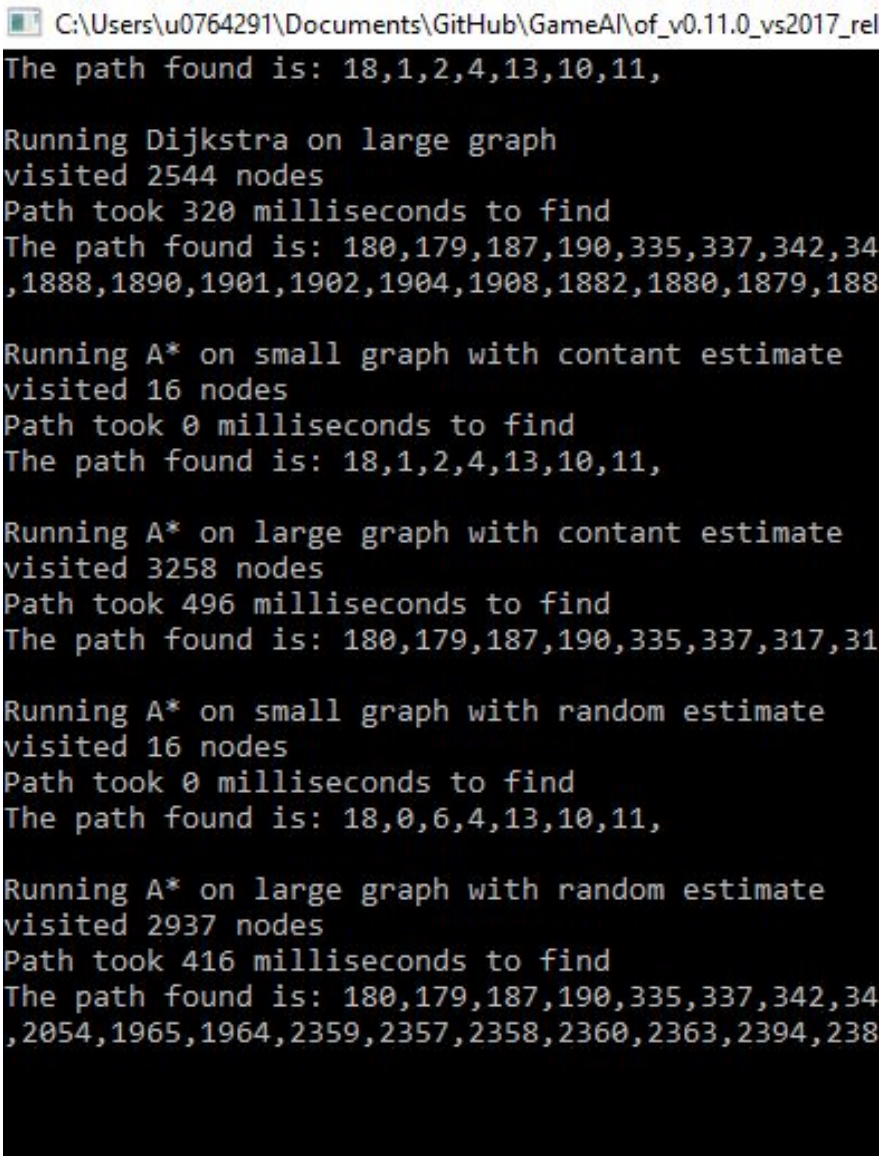

Image E: Dijkstra and A* on small and large graphs time measurements